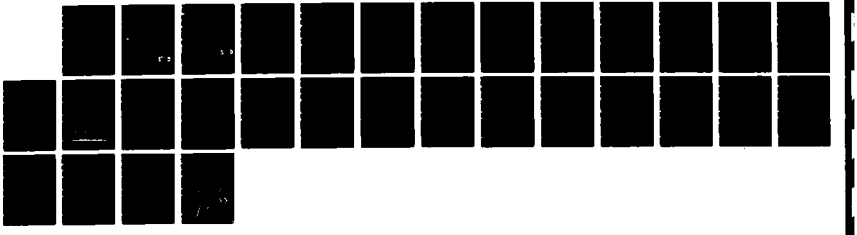
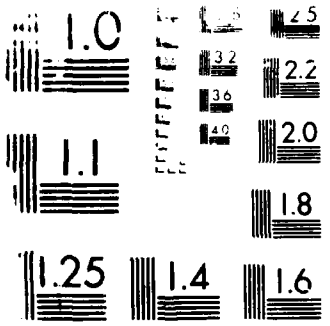
AD-A188 639 EQUIPMENT SIMULATION FOR LANGUAGE UNDERSTANDING(U) NEW 1/1
YORK UNIV NY COURANT INST OF MATHEMATICAL SCIENCES
T KSIEZYK ET AL SEP 87 PROTEUS-M-11 N00014-85-K-0163
UNCLASSIFIED F/G 5/7 NL



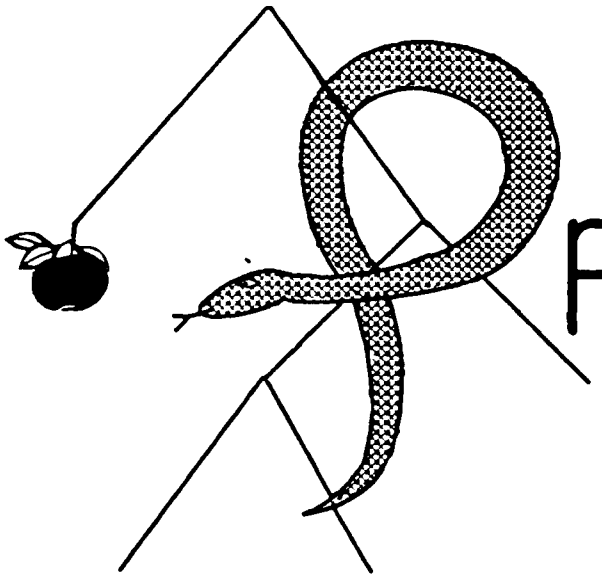

1.0
1.1
1.25
1.4
1.6
2.2
2.0
1.8

(2)

Equipment Simulation for Language Understanding

Tomasz Ksiezyk and Ralph Grishman

PROTEUS Project Memorandum #11
September 1987



PROTEUS PROJECT

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University

2

Equipment Simulation for Language Understanding

Tomasz Ksiezyk and Ralph Grishman

PROTEUS Project Memorandum #11
September 1987

This report is based upon work supported by the Defense Advanced Research Projects Agency under Contract N00014-85-K-0163 from the Office of Naval Research, and by the National Science Foundation under grant DCR-85-01843.

Equipment Simulation for Language Understanding

Tomasz Ksiezyk and Ralph Grishman

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
New York, New York 10012
(212) 998-3491, ksiezyk@nyu-csd2

Abstract

We consider in this paper the task of analyzing reports regarding the failure, diagnosis and repair of equipment. We show that a general knowledge of equipment is not sufficient for a full understanding of such reports. As an alternative, we describe a system - PROTEUS - which relies on a detailed simulation model to support language understanding. We describe the structure of the model and emphasize features specifically required for language understanding. We show how this model can be used in analyzing and determining the referents for complex noun phrases. We point out the importance of identifying the implicit causal relations in the text and show how the simulation capabilities of the model support this task. Finally, we indicate how PROTEUS may be extended to facilitate the entry of new equipment models and to assist in equipment diagnosis.

1. Introduction

The work presented here is part of PROTEUS (the PROtotype TExt Understanding System), currently under development at the Courant Institute of Mathematical Sciences, New York University.[1] The objective of our research is to understand short natural language texts about equipment. Our texts at present are CASualty REPorts (CASREPs) which describe failures of equipment installed on Navy ships. Our initial domain is the starting air system for propulsion gas turbines. A typical CASREP consists of several sentences, for example:[2]

> Unable to maintain lube oil pressure to SAC [*Starting Air Compressor*]. Disengaged immediately after alarm. Metal particles in oil sample and strainer.

It is widely accepted among researchers that in order to create natural language understanding systems robust enough for practical application, one must provide them with a lot of common-sense and domain-specific knowledge. However, so far, there is no consensus as to what is the best way of choosing, organizing and using such knowledge.

The novelty of the approach presented here is that, besides general knowledge about equipment, we also use a quite extensive simulation model for the specific piece of equipment which the texts deal with. We see the following merits of having a simulation model:

---

[1] This work is being done in collaboration with the Unisys Defense Systems. as part of the DARPA Strategic Computing Program.

[2] Unless noted otherwise, all examples given in this paper are from actual CASREPs.

(a) The model provides us with a reliable background against which we can check the correctness of the understanding process on several levels: finding referents of noun phrases, assigning semantic cases to verbs, establishing causal relationships between individual sentences of the text.

(b) The requirements of simulation help us to decide what kind of knowledge about the equipment should be included in the model, how it could best be organized and which inferences it should be possible to make. It appears that the information needed for simulation largely coincides with that necessary for language understanding.

(c) The ability to simulate the behavior of a piece of equipment provides an effective means of verifying the system's understanding of a message: it is relatively straightforward to build a dynamic graphical interface which gives the user insight into the way his input has been understood by the system.

In addition to its role in language understanding, we believe that the simulation model provides many -- perhaps most -- of the capabilities which would be needed for a model-based diagnostic or tutorial system.

We can compare our approach to model building with that often followed in application systems with natural language front ends, such as data base, diagnostic, and tutorial systems. In most cases such front ends were conceived as add-ons to existing systems, often only after such systems were in use for some time. This leads to a situation where much of the knowledge necessary to solve some understanding problems already existed in the main system, but couldn't be used by the natural language module because its structure had been designed without regard to the needs of language analysis. As a result a separate knowledge base had to be built for the language module, duplicating to a large extent the knowledge in the base system, and the two knowledge bases had to be interfaced, complicating the overall design.

We have in contrast focussed from the outset on the needs of natural language understanding. We have been led to develop a detailed simulation model, as would be required for a model-based diagnostic or tutorial system. We have developed a rich graphical interface, which would be of benefit for these other applications too. But we have also incorporated features which were specifically dictated by the needs of language understanding, such as mixed static/dynamic model and the ability to refer to dynamically-created aggregates. We believe that this conceptually richer model, suitably extended, would provide a good basis for a model-based expert system with an *integrated* natural language interface.

In a language system incorporating such a simulation model, the analysis of a piece of text involves the following steps: (1) locating in the model the objects and other concepts mentioned in the text (they are usually referred to by noun phrases) and creating internal representations for those objects and other concepts which are referred to in the text but are not part of the model; (2) recognizing elementary facts reported in the text and building their representations; these facts are either about the above objects and concepts or about other facts; and (3) finding relationships between elementary facts which make them into a coherent whole; such relationships may be given explicitly in the text or may have to be inferred; the inference mechanism is based largely on queries to the model and on requests for the simulation of certain facts.

In the remainder of this paper we shall address the following issues: Why is a detailed equipment model needed? How should such a model be structured and, in particular, what balance should we strike between a static model and one created dynamically as the text requires? How should a noun phrase analyzer be organized to utilize such a model, and how is it affected by this static/dynamic balance? How does the discourse analysis module work and what role does the simulation play in it? What are the ways of supporting users in designing new equipment models?

2. The Need for a Model

In many natural language understanding systems the knowledge about the domain of discourse is organized in the form of prototypes for objects and actions, and for the relations between them which are relevant for the domain. The prototypes are repositories for general knowledge about the instances they may subsume. As each sentence is processed, it is split into units which are identified as possible instances of prototypes in the knowledge base. Through these prototypes access to general information about the concepts invoked by the sentence is achieved. This information is often necessary for the adequate interpretation (i.e. understanding) of the sentence. To account for the fact that the understanding of an utterance depends sometimes on context, it is necessary to maintain information about the discourse context. One way of organizing this information is by creating and storing instances of prototypes for entities from the text as they are analyzed. The combined information coming from the context and from the processed sentence is used to solve problems such as anaphora resolution, connectivity, etc.

Assuming this approach, let's consider the task of analyzing a text beginning with the sentence[3]:

Starting air regulating valve failed.

Having completed the syntactic and semantic analysis of the sentence, we would recognize *starting air regulating valve* as an example of the prototype *regulating valve*. We would then fetch its description and create an instance of a regulating valve. Next, using the general knowledge about valves (of which regulating valve is a more specific case), and the semantic information about *starting air*, we would modify the just created instance with the fact that the substance the valve regulates is starting air. From the syntactic analysis we would know that *starting air regulating valve* is the subject of the verb *fail*. Using the prototype of the action *fail*, we would create its instance and possibly also further modify the valve instance so that its operational state is recorded. These two instances would now constitute the discourse context so far. Now, suppose the message continues with the sentence:

Unable to consistently start nr 1b turbine.

The processing would be similar to what has been described above for the first sentence. We would create an instance of a gas turbine, would fill its proper name slot with *nr 1b* and finally use the instance as an argument in another instance recording the finding about start problems.

These two sentences come from an actual CASREP. In the starting air system (our initial domain) there are three different valves regulating starting air. Two questions, crucial for understanding the report, might be posed in connection with this short, two-sentence text: (1) which of the three valves was meant in the first sentence? (2) did the reported difficulty with starting the turbine mean that the turbine itself was impaired in some way and should thus be included in the report summary, along with the valve, in the list of damages?

General knowledge of equipment may tell us a lot about failures, such as: if a machinery element fails, then it is inoperative, or if an element is inoperative, then the element of which it is a part is probably inoperative as well, etc. Unfortunately, such knowledge is not enough: there is no way to answer these two questions without access to rather detailed knowledge about how various elements of the given piece of equipment are interconnected and how they work as an ensemble. In our case we could hypothesize (using general knowledge about text structures) that there is a causal relationship between the facts stated in the two sentences. To test this hypothesis, we would have to consider each of the three valves in turn and check how its inoperative state could affect the starting of the specific (i.e. nr 1b) turbine. To perform these tests we would need a simulation model. If one of the three valves, when inoperative, would make the turbine starting unreliable, then we could claim that this valve is

[3] We should point out that in this domain *starting air* is a type of *air*.

the proper referent for the *starting air regulating valve* mentioned in the first sentence. This finding would allow us to answer question (2) as well. In the case where the failure of the valve is the cause of the trouble with the turbine, it is plausible to assume that once the valve is replaced, the turbine would also function properly. Hence, it shouldn't be reported as damaged.

Another important argument in favor of an equipment model is related to the interpretation of noun phrases (NP). One notable feature of technical texts is the heavy use of nominal compounds. It seems that their average length is proportional to the complexity of the discourse domain. In the domain of the starting air system, examples like

stripped lube oil pump drive gear

are, by no means, seldom occurrences.

The problem with nominal compounds is their ambiguity. Syntactic analysis is of almost no help here. Even using semantic (selectional) constraints, as in [Finin 1986], substantial ambiguity often remains. When we know that the nominal compounds refer to objects existing in the system, and we have access to a model of the system, we can impose much tighter constraints, thus reducing the ambiguity. Model-based causal reasoning (as illustrated above) may reduce this ambiguity further. The problem may be metaphorically described as a jigsaw puzzle: given several pieces (concepts corresponding to individual words from a nominal compound) put them together to build a sensible picture (concept corresponding to the entire nominal compound). The task becomes somewhat easier in cases when we know that nominal compounds refer to concepts existing in the system. In terms of our metaphor this translates into a hint: a set of pictures is given with the assumption that the solution is one of these pictures. Our model plays a role of such a set of aiding pictures.

The above two considerations demonstrate that in cases where the domain is very specialized and complicated (a typical situation for real-life equipment), language understanding systems should be provided not only with general knowledge about the equipment but also have access to its model.

3. Related work

There are two areas of AI research to which we can relate our work. The simulation model can be considered as an exercise in qualitative physics. The discourse analysis module can be reviewed as a contribution to the natural language understanding of narratives.

A good collection of articles on commonsense reasoning about physical systems was published in 1984 as a special issue of *Artificial Intelligence*, reprinted as [Bobrow 1985]. The major contributions to this volume by [Forbus 1984], [de Kleer 1984], and [Kuipers 1984] as well as some other works published elsewhere, like [Bylander 1985] give a good picture of the field. The approach we took for simulation shows some similarities with this research. However, because our main objective is language understanding, we didn't investigate the issue of naive physics at the depth demonstrated in the cited papers. On the other hand, the demands of discourse analysis lead us to some interesting issues not addressed before. [de Kleer 1984] comes closest to our simulation model. We also use a device-centered ontology in which the devices like valves, filters, etc. play the role of the primitives. The *no-function-in-structure* principle, *qualitativeness* (the variables used to describe the behavior of the device can only take on a small predetermined number of values), and the *quasistatic approximation* are central to the PROTEUS simulation. Similarly, in the equipment simulation we rely on three physical constituents: materials (in our case working substances), components (in our case equipment units), and conduits. The major difference is that instead of a sophisticated qualitative calculus (based on *confluences* in [de Kleer 1984]), we use a much simpler approach in the form of simple rules, defined for the equipment units. which map inputs to outputs (simulating feedback was not necessary for our purposes). In terms of the distinction between structural, behavioral, and functional descriptions [Kuipers 1984], we deal with the first two only. Another simplification is our treatment of processes as events for example, we simulate *corrode* as an event which occurred at a certain moment.

The most difficult challenge, not present in the research on naive physics, followed from the impossibility of including in the model everything which could possibly be mentioned in the CASREPs (we discuss this in more detail in section 5.5). This required us to provide a means for modifying the model dynamically (i.e. during text processing), both by adding new components and by imposing new structures on the existing parts. The compositionality principle proved very useful for solving the latter problem. Another interesting issue corresponds to the recognition of objects on the basis of the quite long and intricate noun phrases encountered in the reports (see section 6). Some parts of the structural and behavioral descriptions had to be provided especially for this purpose. Particularly more difficult was the problem of combining both types of information, that used for recognition and that for simulation, into one. Still another novelty was the use of generic structures (see section 5.2), which were developed for the purpose of keeping the size of the model as small as possible.

One of the earlier approaches to understanding narratives was based on story grammars [Rumelhart 1977] which attempted to specify general structures common to all stories of a particular type. This approach was limited in the range of stories it could handle and, because of its top-down character was ill-suited for unexpected input.

An effective solution to understanding texts in restricted domains is based on *information formatting* [Sager 1978], which uses a classification of the semantic relationships within the domain to derive a tabular representation of the text. Because of the attractiveness of this approach for the specialized domains, it was elaborated in [Marsh 1984] for processing CASREPs: a production system was used for interpreting the information in the table created by the information formatting. It was possible to generate short one-line summaries for the reports. For longer and more complicated reports this method was insufficient.

Much recent work on understanding texts stresses the importance of providing a solid basis of prestored knowledge necessary to interpret text fragments and to link them into a coherent whole. This knowledge is usually organized in the form of pieces of stereotyped information; the two most widely used mechanisms are *frames* and *scripts*. The approaches based on scripts are based on *conceptual dependency* theory proposed by Schank. One of the characteristic features is their relative neglect of syntactic information. The parsing of a sentence is concept driven, aimed at producing conceptual dependency structures. Scripts account for stereotypical information only. To deal with novel situations the concept of a *plan* was introduced [Schank 1977] and essentially elaborated by [Wilensky 1983]. Plans are primarily used for problem solving in situations where one has to satisfy a given goal. In understanding, plans are used in a different way: they help to follow narrated goals and actions of participants in the story in order to make inferences essential for understanding. [Lehnert 1982] extends these ideas to larger patterns.

The research based on frames (e.g. [Bobrow 1977b]) usually assumed a syntactic analysis of text as the first stage in the understanding process. Otherwise, the crucial problems were similar: how to organize knowledge (e.g. declarative vs. procedural), how to identify the appropriate frame (script) relevant to the analyzed fragment, what to do in case of wrongly chosen candidates, etc. Because, typically, the stories dealt with human actions, the most difficult aspect they presented was the variety of possible plots. This necessitated maintaining a large collection of frames which added to problems on every stage of processing. Frames were also successfully used for building more specialized understanding systems, like the PAINTING program [Charniak 1978] which was designed for understanding stories about "mundane" painting. One of the assumptions made there was that a program which is expected to understand texts about a certain type of activity should be also able to perform this activity. We embraced this conviction, believing that PROTEUS will be able to understand messages about equipment damage, if it can simulate the equipment's behavior. Otherwise, the approaches used for PAINTING and for PROTEUS are quite different. We are not aware of any text understanding system which relies on simulation to a comparable extent. There are similarities with other approaches (e.g. [Bobrow 1977a]), however, in the way the

structural knowledge is represented: we use a frame-like approach based on the *Symbolics* flavor system.

4. PROTEUS structure

Fig. 1 presents a high-level flowchart of PROTEUS, indicating how the model is used in the understanding process. [Grishman 1986] describes the overall organization of PROTEUS in some detail.

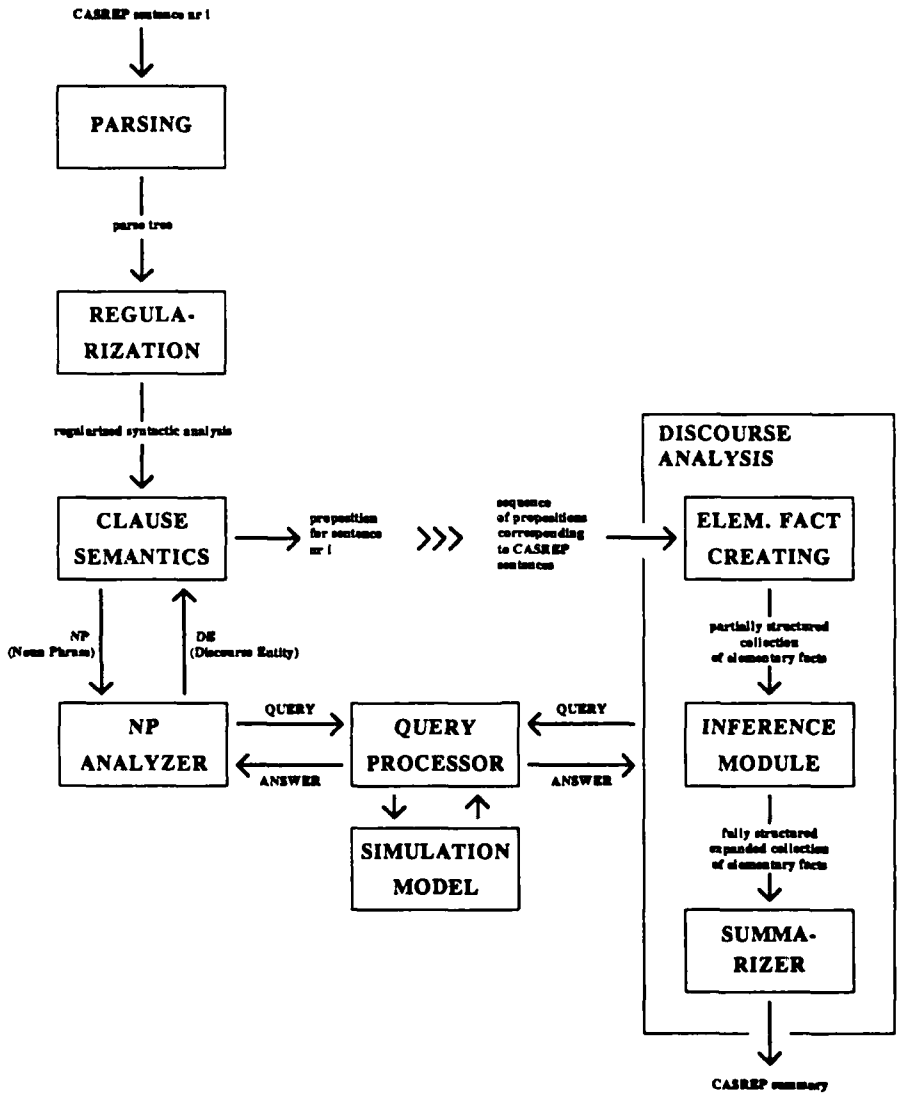


Figure 1. PROTEUS structure.

PROTEUS processes sentences sequentially (first syntax, then semantics, finally discourse). Each sentence is first parsed using a augmented context-free grammar; the parse trees are then regularized, mapping all the different clause structures into a standard verb-argument form. These regularized syntactic structures are then fed to semantic analyzer.

Each sentence is converted by
Clause Semantics into an atomic proposition or several propositions connected by logical, modal, epistemic, or temporal operators. An atomic proposition consists of a predicate and arguments which are either discourse entities determined by the **NP Analyzer** or other propositions. The design of clause semantics is based on Inference-Directed Semantic Analysis [Palmer 86].

Once the syntactic and semantic analysis of all the sentences from a CASREP is finished, the created propositions are fed to the **Discourse Analysis Module** as a sequence in the same order as the corresponding sentences in the report. Each proposition is first translated into some number (possibly 0) of "elementary facts". These elementary facts can be connected with links of various types representing temporal, causal, and other relations. These relations are partially derived from the operators in the representation generated by the **Clause Semantics Module**. Next, the **Inference Module** tries to infer implicit relations, thus augmenting these links, and if necessary expand the collection of elementary facts by inferred and hypothesized ones. This enriched structure of elementary facts constitutes everything PROTEUS was able to understand from the analyzed report. Depending on the particular application, a summary is extracted from the output of the **Inference Module**. All parts of the **Discourse Analysis Module** as well as the **NP Analyzer** make use of the information provided by the **Simulation Model** in the form of answers to queries.

5. PROTEUS Simulation Model

5.1. Characteristics of the domain

The target domains for PROTEUS are *equipment units* (EU): complex technical systems which accomplish physical tasks on demand. These tasks are carried out as serial and parallel combinations of simpler tasks, which are performed by constituent EUs of the main equipment unit. Often these simpler tasks can be decomposed further, leading to a hierarchy of tasks and EUs.

The EUs transmit their effects through various *media*, such as gases, liquids, mechanical movement, and electric current. These media travel from one EU to another through *conduits* appropriate to the different types of media.

Most complex technical systems provide a monitoring capability, using instruments which display properties of the media at various points. They also provide control signals by which the system operator can control the equipment.

5.2. The structure of the simulation model

PROTEUS uses the model in several different ways:

(1) The model is a repository for structural information about the equipment. This information is used by the NP Analyzer both for determining the referents in the model for NP words and for checking relations between them as indicated by other NP words (e.g. for finding out whether *pump* and *gear* are related by the *drive* predicate). The structural information is also used during discourse analysis: for finding the equipment elements which fulfill certain conditions (e.g. for hypothesizing new facts, see below) or for checking relations between elements (e.g. for deciding whether a given IF-THEN rule capturing general knowledge about technical equipment is applicable).

(2) The model is a vehicle for the equipment simulation which in turn is the main mechanism for making inferences during discourse analysis.

(3) The model contains also information necessary for graphical display.

PROTEUS models have the structure of recursive transition networks. They consist of *nodes* connected by directed *links*. The nodes correspond to the constituent EUs of the system; the links to the conduits connecting the EUs. The hierarchical structure of the EUs is reflected in the hierarchical structure of the networks. To represent the internal structure of an EU, we have the corresponding node point to another network in the model. Such (non-

terminal) nodes are called **system nodes**. This recursive refinement must stop somewhere; the (terminal) nodes - which don't have underlying networks in a given model - are called **simple nodes**. We shall use the term **model-networks** to refer to the entire collection of networks modeling a piece of equipment.

Associated with each link is a **working-substance** (WS). These WSs correspond to the media entering and leaving an EU (for example, the rotary motion provided to a pump and the fluid entering and leaving the pump). We can think of the WSs associated with links entering and leaving a node as the input and output data of the node.

The nodes, links, and WSs are represented as sets of (attribute, value) pairs. These pairs are used not only to define the system structure, but also to express all other, non-structural information. For nodes and links we call the attributes **roles**; for WSs, we call them **properties**. The attribute values may be pointers to other nodes, links, or WSs, numbers, symbols, or propositions. Some of the values are time-dependent, reflecting the particular state of the model during simulation.

Each node has four (time-independent) roles carrying structural information: PART-OF, STRUCTURE (a pointer to the network subsumed by the node), TO-LINKS, and FROM-LINKS. The current state of a node is recorded in the (time-dependent) roles CONTROL-STATE (a state of the node reflecting the position of some control switches) and READINESS-STATE (a state of the node describing the possible damage reported in CASREPs). These two roles are important for reference and simulation but have no direct impact on the graphics. Some nodes also have (time-independent) roles for simulation parameters, such as TRANSMISSION-RATIO for nodes describing couplings. There are also roles used exclusively for graphics, such as: (time-independent) SIZE and GRID-LOCATION specifying for EUs' icons their sizes and places on the screen and (time-dependent) MOON-POSITION for nodes describing spin elements. This last role keeps the current angular position of the small globe which revolves around the node's icon on the screen to represent rotary movement.

There are two types of links in model-networks: **simple links** which connect two nodes in the same network, and **ancestor links** which incide on a node from a higher level network or exit a node to a higher level network. The two main functions which links play in the model are: to complement nodes in establishing networks and to relate WSs (whose parameters are often mentioned in the reports) to EUs (which also are commonly referred to in the reports) - WSs are associated with links which are in turn connected to nodes describing EUs. The relationship between links and WSs is not one-to-one. For every link there is one and only one WS. For a WS, however, there might be several links all of which are associated with the WS. This is so because the model might have representations on several levels of detail. For example, there might be a link between a diesel and a transmission system on one level of representation and another link between an adapter hub of the diesel and a driving shaft of the transmission system on a lower level. For both these links there should be a common WS.

Each link has the (time-independent) roles: WS-PTR (working substance associated with the link) and TO- and FROM-END (link's end and start nodes). The time-dependent aspects of links are important for display purposes only and are derivatives of time-dependent attributes of the WS associated with them. The graphical display for links is much more complicated than that for nodes because links are the main vehicle of the dynamic graphical interface of PROTEUS: link icons are animated, reflecting such features of the working substances as flow and pressure in the case of liquids and gases or speed in the case of rotation. Some of the link's roles serve this animation, e.g. JERK-PTR gives the coordinates of the bar which is to be jerked next in the animation of air flowing through a link's icon.

Each WS has a (time-independent) attribute LINK-PTR which takes as value the set of all the links which point to the WS. The time-dependent properties of WSs used for the simulation vary from case to case depending on media: for liquids they may be: PRESSURE, TEMPERATURE, FLOW, COLOR, and PURITY; for rotary movements: ROTARY SPEED.

Usually, some of these properties have their correspondents used for reference purposes. For example, for liquids we may have NORMAL-OPERATIONAL-PRESSURE or NORMAL-OPERATIONAL-TEMPERATURE. This is necessary because otherwise it would be impossible to find a correct referent for a noun phrase like *high pressure oil pump* in a state of the model in which the pump is not working. Modifiers of this kind usually refer to a situation of normal operation of the equipment: independent of whether this pump is running or not or whether - because of a failure - the oil it disgorges has only low pressure, it can always be referred to in the reports as a *high pressure oil pump*. For a more detailed discussion of this example see the section on noun phrase analysis.

Usually, a model network is subsumed by only one system node. However, in order to avoid unnecessary escalation of a model's size, we allow a system network to be subsumed by several system nodes. We call such networks **generic**. This mechanism is useful for EUs some of whose components are identical. For example, in our domain of the starting air system, there are three identical air compressors each of which requires over 100 nodes. Generic networks allow us to avoid this costly repetition by modeling an air compressor only once and for each of the compressors having a node whose STRUCTURE attribute takes the same value, namely the pointer to the generic network. Two modifications are necessary in order to make generic networks work. Firstly, the TO-ENDs of a generic network's exiting ancestor links and FROM-ENDs of a generic network's entering ancestor links can no longer be individual nodes on a higher level; for each ancestor link in a generic system there will be more than one way of ascending to a higher level (either forwards using the TO-ENDs or backwards using the FROM-ENDs). In fact there are as many different ways to do this as there are system nodes subsuming the generic network. Secondly, all the time-dependent attributes of the nodes, links, and WSs of a generic network will have distinct values for each system node subsuming the generic network. One straightforward solution to these problems, taken in PROTEUS, is to use as values of the attributes mentioned above, arrays of size equal to the number of system nodes subsuming a given generic network. With this approach an EU which is modeled by a generic network is fully described by this network and an array index.

5.3. The simulation process

As we indicated at the beginning of this section, one of the main purposes for which we intended our model is the qualitative simulation of the modeled EUs (in most cases a precise quantitative simulation is not required for language understanding). Simulation is performed by an event-driven algorithm which is triggered by an external event (such as operator action or reported failure) and continues until a stable state is reached. Although it is possible to have several levels of detail in the model, the simulation is always conducted on the lowest level, i.e. on the level of simple nodes. To each of these nodes a **model function** is assigned. A model function of a node consists of two parts:

(a) for any possible change in the node's readiness state, it defines the changes of WSs associated with its input and output links;

(b) for any possible qualitative change in any of the WSs associated with the node's input and output links and for any possible values of the readiness and operational state attributes, it defines the changes of the WSs corresponding to all other input and output links.

It should be noted that a change in the operational state of a node is expressed by a change in the signal WSs associated with some input link of the node. EUs can be divided into two groups depending on how they are controlled: those in which a certain state is maintained as long as some input WS has a certain value, e.g. a solenoid stop valve remains open only as long as the electric signal to the valve is maintained (nodes describing such EUs in our model don't need an OPERATIONAL-STATE attribute at all); and those where the input control signal is needed only to set the EU in a new state, e.g. a diesel which is turned on and off by means of a switch (here the OPERATIONAL-STATE attribute is necessary).

The simulation can be viewed as an interweaving of intervals of stable and unstable states

of the model. An unstable phase is started by an external event such as a control action or damage to some node of the model. Once it happens, the model function of the node is applied and all changes in the WSs associated with the node's input and output links are determined. These changes propagate (forwards through output links and backwards through input links) to the neighboring nodes. Each of the nodes affected in this way is considered in turn and its model function determines how the changes propagate further. This process continues until there are no more changes. At this point simulation enters its stable phase and remains in this state until another external event occurs.

One feature of the model functions is that there are no delays in propagating the changes triggered by an external event. However, some of the analyzed messages mention (explicitly or implicitly) such delays, e.g. the time gap between the act of starting a turbine by an operator and the time when the turbine gets actually started may extend up to several minutes. To accommodate such cases we introduced **demons** as a third kind o. vent starting an unstable stage of the simulation. For demons to work a simulation clock is required. A demon consists of two parts: a time point when it should be activated and a demon procedure name. Demons are set during the execution of the model functions of nodes. A demon procedure is executed when the clock time reaches the demon activation time. The procedure determines what changes should be made to the WSs at the input and output links and possibly to the operational state of the node. The propagation of these changes proceeds like those for control activities and damages.

None of the messages we analyzed required simulation of feedback. Therefore, the PROTEUS simulation was not designed to deal with feedback. Considering the possibilities of using PROTEUS-like models for applications other than language understanding, notably diagnostic and tutorial systems, this must certainly be viewed as an important limitation. The research on *incremental qualitative simulation* [de Kleer 1984] gives reasons to believe that extending the coverage to feedback is possible in the framework of our approach to simulation.

5.4. Generality of the model

We tried is to design PROTEUS in such a way that it may be adapted easily to new equipment. Clearly, the model has to be built anew each time we want to use PROTEUS for a new equipment unit. To minimize the work required, we distinguish between knowledge about equipment in general and knowledge about a specific piece of equipment. We tried to achieve this duality of knowledge using prototypes and their instances: the model would be built of instances of prototypes. The prototypes constitute part of the general knowledge data base. In the instances we store only the information which is specific to the object described by the instance. For example, in the case of a gearbox, the information about its function (speed change) should be stored in the prototype, and only the ratio of this change should reside in the instance of a specific gearbox. Also the information about how a specific gearbox is used in the domain equipment must be kept in the instance. Of course, the prototype-instance scheme ensures that all the general knowledge connected with the prototype is also accessible from instances of this prototype. We found the rich repertoire of programming tools constituting the flavor system in *Symbolics-Lisp* a very convenient vehicle for implementing this strategy.

5.5. Level of detail

In the preceding section we have provided a mechanism for a stepwise recursive refinement of the model. The EUs we want to model are both big and complex. For example, our initial domain, the starting air system, is described in the ship's manual on 28 pages of text, figures and tables. We need some kind of simplification. A possible approach would be to refine the hierarchy far enough so that everything which potentially may be referred to in the reports would have a description in the model. This, however, seems impractical. Consider a typical sentence from one of the reports:

Investigation revealed a broken tooth on the hub ring gear.

Considering that there are several different gears in our starting air system and each of them has many teeth which are very much alike, it's obvious that creating a separate description for each of them wouldn't be reasonable. The same remark is true for balls in bearings or for connecting elements like screws, bolts or pins. On the other hand, information about the tooth conveyed in the above sentence cannot go unnoticed. The solution we accepted for such elements is not to include their descriptions in the model on a permanent basis but to keep open the possibility to create and add them to the model if such a need arises during the analysis. A rule of thumb for deciding whether a particular element deserves a permanent place in the model can be formulated as a question: How much information specific to this element is necessary to solve understanding problems, like finding referents (see the section on nominal compounds) or making inferences? As an example of the latter, let's consider a specific gear. We would like to know, among other things, what is this gear's role and place in the modeled equipment so that, in case of its damage, we could determine the impact on the equipment. Information of this type can be deduced neither from the analyzed text nor from general knowledge about gears. It must be known in advance. Our way to achieve this is to keep the gear's description permanently in the model.

There are, however, elements like teeth which have so little relevant structure that they are always referred to as *tooth, teeth* together with the element higher up in the part/whole hierarchy (let's call such an element a *host*). Thus, it is not necessary to maintain any specific information about them in the model. It will suffice, if we are able to create their descriptions when they occur in the text. All the possible information we will ever need to include in such descriptions will come from the text. The information relating such elements with other parts of the equipment will come from their hosts. For example, the impact of a tooth's damage on the equipment may be derived from the functional information connected with its host.

We will return to this issue of statically and dynamically modeled objects later in this paper.

It is important to notice that there is nothing absolute in distinctions such as the one made above. It is conceivable to have a piece of equipment of a larger scale than the SAC, where elements like gears are not essential enough for us to be bothered with their shapes or locations; if broken they probably would be referred to by giving the higher-level element of which they are part. In such cases we would rather treat gears like we treat teeth here.

For simulation alone, it would be sufficient to keep in the model only those networks which are on the deepest level. However, because the texts refer to equipment parts at various levels of detail, we have to maintain all intermediary levels of the model. Furthermore, because of the requirement of relating facts about such parts by means of causal links, we should also be able to cross levels in traversing the network. Such traversals are possible using the **ancestor links**.

5.6. An example: the starting air system model

As our initial domain we have chosen the "starting air system" used for starting gas turbines on Navy ships. The model consists of 15 networks with a total of about 175 nodes.

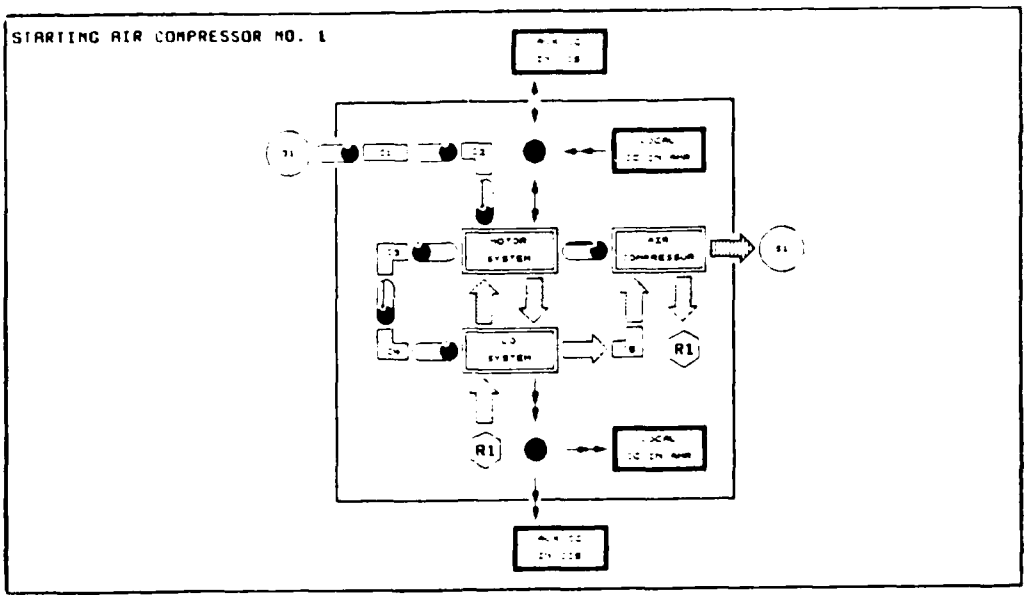


Figure 2. Model network for SAC.

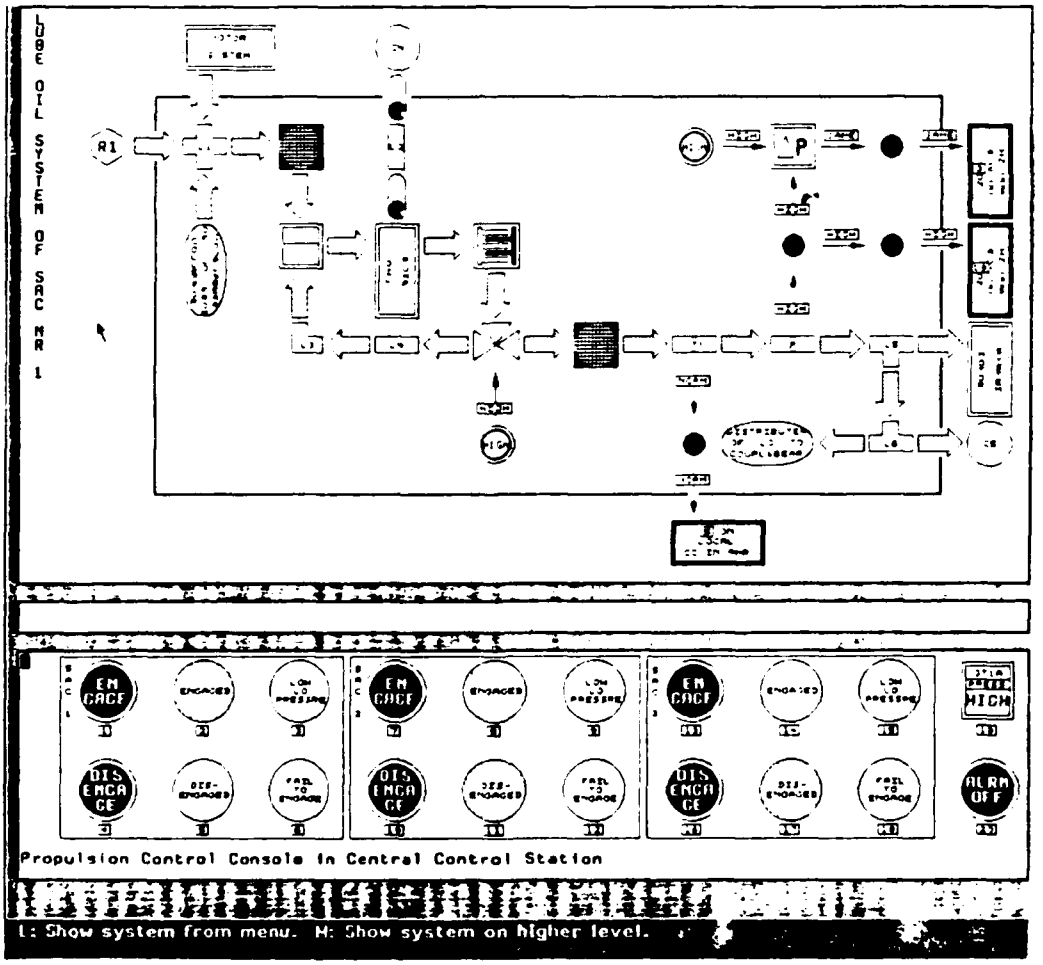


Figure 3. Model network for SAC LO System.

We show here two networks and one control console of the model. Both figures are *Symbolics* screen images generated by PROTEUS from the model networks. Fig. 2 is the starting air compressor (SAC) network. The rectangles enclosed in double lines are the system nodes, which point to more detailed networks in the model. One of these is the LO (lube oil) network, shown in Fig. 3. Simple nodes are represented by icons, such as the filter, cooler, and valve in Fig. 3. The rectangles enclosed in heavy black lines represent control consoles from which the system may be operated and monitored. Clicking the mouse on the ON or OFF buttons turns on or off the corresponding EUs. The control consoles also have indicator and alarm lamps as well as instrument displays. The arrows represent icons of links associated with the flow of WSs like gases and liquids. The insides of these arrows show some of the parameters of these WSs. For examples, the bars indicate air and the dots oil. The density of these bars or dots corresponds to the pressure of air or oil, respectively. When the bars or dots remain stationary, it means there is no flow in the conduit corresponding to the link. The rods represent icons of links associated with rotation WSs. The rotation is indicated by small filled-in globes (moons) on both ends of a rod and a line connecting these globes. The speed with which these globes rotate indicates the speed of the corresponding WS. These dynamic displays provide a direct visual presentation of the system's understanding of a message (oil may stop flowing or a gear rotating). They are achieved as a side effect of the simulation used for understanding purposes. The links crossing the outline boundaries are ancestor links (links which connect the network to other parts of the model); the other arrows are simple links.

6. Noun Phrase Analysis

6.1. The role of noun phrase analysis

The goal of the noun phrase analyzer (NPA) is to convert a noun phrase into a set of referents that may be used by subsequent stages of the system. For example, for a noun phrase describing an EU, the NPA would return a pointer to the corresponding node in the equipment model.

Why is this a difficult task? As we noted earlier, the texts in this domain are replete with long compound nominals, such as *stripped lube oil pump drive gear*. Parts are not always referenced by the same name, even within a single manual (for example, *surge control system* and *compressor surge valve assembly* refer to the same EU); parts may be described by position, function, or other properties. Consequently, we cannot simply use a large term dictionary; we must analyze the structure of each noun phrase. Specifically, we must determine what modifies what within the noun phrase; for example, that "lube" (short for "lubricating") modifies "oil" rather than "pump", "drive", or "gear". We must also identify implicit relations among elements; for example, that "gear" is *an adjacent part to* "pump". Syntax is of little help in this task, so the PROTEUS syntactic analyzer passes prenominal modifiers through as unanalyzed strings. It is thus left to the NPA to determine the structure and implicit relations of the noun phrase.

The equipment model serves two functions for the NPA: First, the model is used to confirm possible relations between noun phrase constituents (for example, that there is a gear which is adjacent to a pump). Second, once a structure has been determined for the noun phrase, the model can be used to provide a set of referents for the phrase. We call these referents **discourse entities** (DEs).

The interaction between the NPA and the model is mediated by the **Model Query Processor** (MQP). The MQP evaluates domain predicates relative to the equipment model, and creates DEs when needed. In this way the NPA need deal with only two levels of representation, the linguistic representation and the representation in terms of domain predicates, and is made independent of the details of the equipment model representation.

6.2. Word classes

For the purposes of the noun phrase analyzer, the words which may occur in noun phrases are grouped into *semantic classes*. These classes reflect the ways in which the words may combine to form larger phrases. They are specific to the equipment domain but are independent of the specific model representation.

Each noun phrase (and each constituent phrase within a noun phrase which is identified by the noun phrase analyzer) must be assigned an internal representation for use in subsequent processing; this representation is the *discourse entity* mentioned above. Different types of entities require different internal representations, so the entities are grouped into *discourse entity classes* to reflect these different representations. These classes are described below (section 6.4.1).

Words which cannot stand alone as noun phrases, but only participate in noun phrases as names of relations (e.g., *regulating*) or as arguments of relations (e.g., adjectives of shape) will not have corresponding discourse entities. Thus these words will belong to a semantic class but not a discourse entity class.

6.3. The Analysis Procedure

The NPA begins by fetching (from the dictionary) the semantic class of each word in the noun phrase. Constituents are combined bottom-up based on a set of rules stated in terms of these classes. These rules identify relationships of the form

(Pred Arg-1 Arg-2 ...)

where *Pred* is a predicate of the domain and the *Arg-i* are constituents of the noun phrase. From a detailed analysis of part of our corpus we have identified a set of 16 predicates for analyzing noun phrases:

adjacent-to alarm assemble couple drive location lube made-of
measure name operate-on part-of regulate saturate shape start

Most current systems validate the application of these rules through selectional checks (constraints on the argument classes of each predicate) [Finin 86]. We perform such checks and then go a step further and check for the existence of the specific relation between specific entities. This is done through a call to the MQP. The MQP is first invoked for each word to obtain its internal representation in the model, and then for each proposed predicate to verify the existence of the corresponding relation in the model. If the relation exists, the MQP returns the representation for the head constituent of the relation (see section 6.4.3).

Applying these rules to post-nominal modifiers is relatively straightforward: the predicate can generally be determined from the preposition, and the arguments of the preposition are explicit and delimited. After processing the post-nominal modifiers, the NPA turns to the pre-nominal modifiers. Analysis of these is considerably more difficult. The problem is to decide what predicates should be used and what are the arguments of these predicates. Both the predicates as well as their arguments may be given explicitly or implicitly. Examples are: *temperature regulating valve* (the predicate and both its arguments are explicit), *drive gear* (the predicate and one of its arguments are explicit, the other argument, the object of DRIVE, is implicit), *pump shaft* (the predicate, PART-OF, is implicit, both of its arguments are explicit). The NPA considers the semantic features of the items, together with order constraints, to match the items with arguments of some canonical predicate. A match is considered successful if it is possible to identify some (not necessarily all) of the arguments of the predicate among the modifiers. For verification purposes, it is assumed that the empty arguments match anything. Once a matching canonical predicate has been found and as many of their arguments matched with the modifiers as possible, the NPA may pose a verification query to the model.

In some cases, the proper argument to a predicate is an object or concept related to the explicit object. For example, in *air regulating valve* the object of REGULATE is not air itself, but one of the properties of air. A similar case is *high pressure pump*; here, the problem is to find a relation between an EU (*pump*) and a medium property (*high pressure*). To

account for such cases, if none of the canonical predicates match the modifying and the modified items, the NPA repeats the matching procedure with objects or concepts which are variations of these items. Possible variations are: generalizations, fragmentations, specializations (properties for WS, roles for EUs) of the item referents. For example, for the phrase *high pressure pump*, if we fail to find a predicate which relates a pump and a property of a medium, we next try with a pump and a medium which has that property.

6.4. Model Query Processor

The MQP serves both the NPA and the Discourse Analyzer. We will discuss here the queries used by the NPA, i.e. requests for the DE for a word, and predicate verification queries; the queries used by the discourse analyzer are discussed in section 7.5.

6.4.1. Discourse Entity Classes

Noun phrases refer to different types of concepts and so must have different types of internal representation. We have grouped the NPs of this domain into five classes based on the form of the Discourse Entities we create. For some classes the DEs are constituents of the model previously described; for others we dynamically create structures analogous to those in the model. The **discourse entity classes** are:

1. **Equipment Units** (EUs). Used as modifiers (e.g. *compressor* in *compressor shaft*) or as main objects (e.g. *shaft* in *compressor shaft sheared*). Their discourse entities are nodes in **model-networks**. Most of these nodes exist in the model permanently. Some, however, are created during analysis of the NP (see discussion below).

2. **Media.** Used as modifiers (e.g. *starting air* in *starting air compressor*) or as main objects (e.g. *lube oil* in *lube oil contaminated*). Their discourse entities are Working Substances (WSs). All media mentioned in the reports, except those used as modifiers of non-permanent EUs, are assumed to have corresponding WSs in the model.

3. **Stuffs.** Used as main objects only. Their discourse entities are created when a NP describing a stuff is encountered in the text. Their discourse entities consist of three parts: (A) material, (B) form, (C) quantity. Examples are non-metallic particles, chips, metal chunks.

4. **Media Properties.** They are used as modifiers of media (e.g. *high pressure* in *high pressure lube oil*), or represent concepts on their own (e.g. *LO temperature* in *LO temperature increased sharply*). Their discourse entities generally consist of three parts: (A) a WS, (B) name of the property, (C) property value. Not all of these three parts are referred to explicitly in a NP. Possible combinations are: [A, B, C] (e.g. *low output air pressure*), [A, B] (e.g. *LO pressure*), [B, C] (e.g. *high speed*), [B] (e.g. *temperature*). A new discourse entity is created each time a property is mentioned in the text.

5. **Media Property Values.** They are used as parts of 4, above (e.g. *normal* in *normal temperature*), or represent concept on their own (e.g. *zero* in *lo pressure dropped to zero*). They may be numbers (e.g. *zero*), strings (e.g. *(nr 1B)*), or predicates (e.g. *(above (65 psig))*. A new discourse entity is created each time a value is mentioned in the text.

6.4.2. Creating DEs for Words

One issue we have already addressed several times is the mixed static/dynamic characteristic of our representation. The main equipment structure is recorded statically in **model-networks**, while DEs for low-level EUs and words of certain other DE classes are created dynamically. Since the other modules are designed to be independent of the model representation (in particular, the static/dynamic distinction), this distinction must be hidden by the MQP.

We use four different approaches to the DE request queries. The first two are exemplified by the class EU. The third one is illustrated by classes Stuffs and Media Properties and the last one by the class Media.

For permanently modeled EUs, the dictionary contains a set of pointers to all nodes in

model-networks which are its DEs. Such a set assigned to a word is returned if the DE of the word is requested. For example, for the word *pump* the set of pointers to all pumps in the model is returned.

The approach becomes more complicated in the case of DE requests for words which correspond to EUs which are not premodeled. We have two kinds of such EUs:

(1) EUs as parts of other EUs whose DEs are in the model. As indicated in the preceding section dealing with the model, it is impractical to represent every tiny element of the modeled equipment. However, such elements occasionally break and are thus mentioned in the casualty reports. For any such EU we create a new node. The pointer to this node is returned as a result for the DE request. In addition, we must establish a connection between this new node and the rest of the model. This is simple, because the only connection of such a node with the model is through the PART-OF relation with some node in the **model-networks** which is mentioned (explicitly or implicitly) in the text. An example here is: *connecting pin in pump drive assembly*; the connecting pin is not statically modeled.

(2) An EU as a superstructure of some subset of EUs which have been modeled, but not grouped together in a single network subsumed by some node which could be the DE for the EU. The reason for not having such EUs modeled *a priori* goes back again to practicality considerations: the number of different ways in which EUs may be grouped together in a single description is simply too large to be represented explicitly in the model. An example here is: *the coupling from diesel to SAC lube oil pump*. All the spin elements which constitute this coupling have been modeled but are not in a single network subsumed by a node which could have been the DE of this noun phrase. However, for inference purposes and for reference resolution in later parts of the text, this coupling must be identified as a node in the model. A node of this kind has to be created and returned as a result for the DE request. In contrast to the above case, establishing a connection between the new node and the **model-networks** is a complex task involving several input and output links.

Each word of the classes Stuffs and Media Properties has assigned in the dictionary a prototype of a corresponding DE which is instantiated when the DE for a word is requested. The pointer to this newly created DE is returned.

Words of the class Media could be treated like EUs: some of the WSs exist in the model permanently while others have to be created when they occur in the text. We decided not to take this approach here for two reasons: The number of pointers returned would be relatively large (in case of *air*, for example, it would be almost equal to the number of "air" links in **model-networks**). Secondly, in most cases, the modifiers are of no great help in reducing this number. More often, the context and the default values help to disambiguate the reference. Hence, we create for words of this class descriptions in the same manner as for words of classes Stuffs and Media Properties. However, when such a DE is about to be interpreted as part of a predicate, we use it together with the context information to find the correct referent in the model. Consider, for example, the sentence *Air pressure increased sharply*. Analyzing the NP alone, it is not possible to figure out which particular air WS is meant. However, the information about the context allows us to make a reasonable assumption that the right referent is the air WS which is associated with the link entering a pressure measuring instrument.

6.4.3. Predicate verification queries

After a predicate-argument pattern has been identified by the Noun Phrase Analyzer, a predicate verification query is made to the MPQ. The **predicate verification queries** have the form:

(MQ-Pred Head Arg-Name-1 Arg-Val-1 Arg-Name-2 Arg-Val-2 ...)

where *Head* must be one of the *Arg-Name-i*, and the *Arg-Val-i* are DEs returned as result of the DE request queries described above or of predicate verification queries described below. The *Arg-Val-i* are either sets of DEs or individual DEs. Named rather than positional arguments are used because the queries may be invoked with only some of the *MQ-Pred*

arguments. The MQP provides defaults for all the missing arguments.

The *Head* indicates which of the arguments should be modified to be later returned as the query result. We call the Arg-Val of this argument the *head argument*. Depending on whether the head argument consists of premodeled DEs or of dynamically created DEs, two cases are possible:

(1) The head argument consists of premodeled DEs. We treat an individual DE as a one-element set of DEs. The query is then understood as a filter condition. The MQP considers all elements of the head argument set one by one, checking which ones pass the test described by the query predicate and collecting them to form the result to be returned. In the case where any of the other arguments are sets as well, a test will pass if any combination of members from these argument sets makes the predicate true. If none of the members of the head argument set passes the test, the MQP returns NIL, and consequently, the NPA must drop the attempted analysis of the NP from further considerations. For example, for the query

(MQ-Shape :Equip :Shape RING :Equip {GEAR-1 GEAR-2 GEAR-3 GEAR-4})

assuming that only GEAR-2 and GEAR-3 are ring gears, the MPQ returns the set {GEAR-2 GEAR-3}. If none of the four gears were a ring gear, the returned result would be NIL, and the NPA would have to try (if possible, e.g. if the NP were *ring gear assembly* referring to an assembly of gears arranged in a shape of a ring) to consider *ring* as a modifier of some other NP constituent.

(2) The head argument consists of dynamically created DEs (either just one or a set). In this case the query is understood as an additional piece of information about the objects described by the head argument. The MQP updates the DEs of the head argument in such a way that this information is accounted for in the returned DEs. This may be done either by filling some roles in the head argument DEs, or by creating modified copies of them. An example of the former is the query

(MQ-Lubricate :Substance :Substance WS-DESC-7)

which is posed while processing the NP *lube oil*. The returned result is the WS-DESC with its FUNCTION role filled with (LUBRICATE *ANY*). An example of the latter is *compressor lube oil*, for which the query would be

(MQ-Lubricate :Subst :Subst WS-DESC-7 :Equip {COMPRESSOR-1 COMPRESSOR-2 COMPRESSOR-3}

Here, the MQP would return the set {WS-DESC-71 WS-DESC-72 WS-DESC-73}, where the WS-DESC-7i would be modified copies of WS-DESCS-7: their FUNCTION roles would be filled with (LUBRICATE COMPRESSOR-i).

The evaluation of MQ predicates varies from case to case. It is straightforward for *(MQ-Made-Of :Object :Material)*, where only :Object may be specified as a head (*metal particles*), and where it is sufficient to compare the role filler of the tested object with the other argument's value. It is more complicated for *(MQ-Operate-On :Medium :Object)*. Here, both arguments may be used as a head (*oil pump* vs. *SAC oil*). Furthermore, when the object is a system node, it is necessary not only to examine its input and output links, but also - if there is no WS associated with any of these links which has the medium as part of its description - recursively analyze all the nodes beneath the one corresponding to the object.

The evaluation of the MQ predicates may sometimes involve a quite complex analysis of the model. An example of this is *(MQ-Drive :Driving-Object :Driven-Object)*. Not only may both arguments be heads (*pump drive gear*), but because in most cases only one of them is specified explicitly (e.g. *driving adapter hub*, *driven gear*), the default for the missing argument must be determined. The evaluation procedure, in contrast to the previous case, is different depending on which argument has been declared as the head. Let's consider the case when the :Driving-Object is the head. This kind of modifier is used in our domain for EUs which generate or transmit a rotary movement. These EUs are arranged in chains of couplings starting at the source of a rotation and ending at places where the rotation is converted

into some other type of WS. In every coupling from this chain we could distinguish between a driving and driven element. However, in our corpus the only cases where this modifier has been used are: (a) elements which are on the boundaries of higher level system nodes, for example, an adapter hub (lowest level) connecting a diesel with a transmission system (both higher level system nodes); (b) elements which start chains of couplings in networks subsumed by system nodes, for example a shaft which starts a transmission system; (c) special arrangements, for example, *driving* and *driven gears* in a pump. Cases from groups (a) and (b) are resolved by analyzing the structure of the model, whereas cases from group (c) are solved by examining FUNCTION slot fillers of the argument DEs.

6.5. Reference Resolution

If at the end of NP analysis we are left with a set of more than one DE, we have to disambiguate the reference. The reference resolution procedure has four parts.

The first part of reference resolution is invoked as soon as the NPA finishes and consists of the following steps (we call the DEs to be disambiguated *candidates*; they are of a certain type, such as *diesel* or *shaft*):

(1) if any of the candidates is on the focus list (this is a list of the DEs corresponding to all the NPs encountered so far in the analysis of the message) choose this candidate as the intended referent;

(2) otherwise, if the candidates' type is listed on the default table, choose the default DE as the intended referent;

(3) otherwise, mark the ambiguity for later resolution.

This approach follows [Palmer 86].

The second attempt to resolve disambiguity is made by the clause semantics. The approach here uses two sources of information which can assist disambiguation: (a) the semantic constraints imposed on arguments of clause predicates; (b) specific knowledge about the structure of the equipment. Case (b) was illustrated in the preceding section where we have shown how to disambiguate references to *air pressure* (air pressure at any point in the equipment is a possible referent here) in the context of *Air pressure increased sharply*.

If clause semantics fails to resolve the ambiguity, discourse analysis can be of help as illustrated by the example of *starting air regulating valve* discussed in the section on the **Need for a Model**, above.

As a last resort PROTEUS engages the user in an interactive session: an explanatory text is shown in the notification window, the nodes corresponding to the candidate DEs are highlighted on the display one by one, and the user is expected to select one of them with the mouse. The DE corresponding to the chosen node becomes the selected referent.

7. Discourse Analysis

7.1. How much understanding do we need?

One basic objective in understanding the messages is determining the cause of the malfunction. In some cases this information is not explicitly provided. Even reports which have explicit information on equipment damage or malfunction can pose problems. We give two examples:

(1) Several problems might be reported which are causally related to each other, so that fixing the first problem from such a chain is enough. Consider for example the pair of sentences: *Starting air regulating valve failed. Unable to consistently start nr 1B turbine*. The malfunction of the 1B turbine shouldn't lead to the conclusion that the turbine is broken and therefore needs replacement. One should recognize that, once the starting air regulating valve has failed, it is not possible to start the turbine, so probably only the valve has to be fixed.

(2) A report may mention a problem which has been fixed by the crew, so no intervention

from outside is necessary, for example, *Input drive shaft found to be seized.* [4] *S/F [ship's force] reinstalled old SAC utilizing new drive shaft.* These examples indicate the importance of **causal** and **temporal** relationships in understanding some messages.

Another basic objective is understanding the **effect** of the malfunction. This information is crucial for maintaining an accurate picture of the readiness of the ships from which the CASREPs come. It is desirable here to recognize the highest level on which the functionality has been affected. For example, if a CASREP mentions both a problem with engaging a compressor as well as a problem with starting a turbine, it should be possible to recognize that the latter fact gives a more accurate picture of the readiness of the ship. If EU-TOP is the highest-level EU among those whose functionality has been mentioned in a CASREP, then we may have several possible situations:

a) EU-TOP cannot be operated at all,

b) EU-TOP can operate normally, but its redundancy level decreased, e.g. turbine starting became less reliable due to a loss of one of several SACs,

c) EU-TOP operates only in a restricted way, e.g. not consistently.

We are talking here about equipment functionality which has been mentioned **explicitly** in the reports. However, if we broadened the coverage of the model and differentiate among types of damage, it should be possible to infer the effects of the malfunction even in cases where it is not given explicitly.

7.2. The Structure of the Discourse Analysis Results

The structure of the results produced by PROTEUS for a CASREP should be flexible enough to capture all the essential information such as was described in the preceding section. Each CASREP is processed by various Navy organizations from different perspectives. It is therefore natural that the aspects of a CASREP which each place is primarily interested in, differs from case to case. In order to be able to tailor the results to these special needs and at the same time to maintain the highest possible uniformity of the Discourse Analysis Modules across the distributed systems, we decided to produce the results in two stages. We first create a collection of **elementary facts** onto which a structure of causal and temporal relationships is imposed. These results are application-independent and contain everything PROTEUS was able to understand. We then build a **summary** which meets specific needs. The summary is created by extracting from the collection only those facts which are relevant for a given application. We will discuss summaries in more detail below. A user who wants to look beyond the information provided by the summary may access additional information from the collection. PROTEUS provides for this purpose a menu-driven interpreter of queries allowing access to more detailed information about the analyzed report.

Elementary facts describe damages, functionality problems, corrective actions, etc. They are instances of appropriate prototypes designed in a hierarchical manner like the ones for the equipment model. For example, a prototype of a damage is based on the prototype for an event, whereas a prototype for a repair is based on the action prototype. The first elementary facts are created from the set of propositions generated for the CASREP by the Clause Semantics Module. This initial collection may then be expanded by additional elementary facts which are built as a result of inferences and hypotheses. The form of elementary fact instances has been designed so that one can incorporate (as fillers of appropriate slots) various kinds of information which is gained either from the report context (i.e. other propositions) or from prototype defaults. Other slots hold pointers which record casual, temporal, and other dependencies among the facts.

---

[4] This is an inference from a sentence in the CASREP, not an actual sentence.

7.3. Creating Elementary Facts

The input to the Discourse Analysis Module is an ordered set of propositions extracted from the report. Each proposition consists of a predicate and arguments which are either DEs determined by the NP Analyzer or other propositions.

For each predicate there is a prescription telling how a proposition of this kind contributes to the collection of elementary facts (see preceding section). Several interpretations are possible:

(1) A proposition can constitute a basis for creating one or more new elementary facts. The slots of the facts are filled with information coming from the arguments of the proposition, from partial order information relevant for this proposition, or from default values provided by the prototype being instantiated. For each elementary fact there exists a function which allows us to retrieve the proposition which underlies the fact. We need such functions because some of the discourse analysis operations, most notably simulation queries and IF-THEN rules operate on propositions rather than elementary facts (see below).

(2) A proposition can cause an existing elementary fact to be replaced by another one. A common situation occurs when a negation is applied to a proposition which gets transformed into a fact from the class Normal-Op-Events (see below), e.g. *Pump will not turn when engine jacks over*. In this case the *turn* fact is removed from the list of reported facts to be replaced by a fact from the class Malfunction (see below).

(3) A proposition can modify an existing elementary fact. Consider for example the sentence: *Suspect faulty high speed rotating assembly*. The *suspect* proposition is not recorded as a separate elementary fact but rather as part of the elementary fact describing the damage of the high speed rotating assembly, in the slot which indicates the degree of certainty of the fact.

(4) A proposition can convey information relating other elementary facts. Consider the sentence *Borescope investigation revealed a broken tooth on the hub ring gear*. Elementary facts are created both for the *investigation* and for the *break* proposition. However, the meaning of the *reveal* proposition can be restricted to its function as an indicator of how these two elementary facts are related: again, viewed from the perspective of understanding it seems sufficient to interpret *reveal* by establishing an EVIDENCE link between the two facts. Another example: *Hub failed, causing spline assy to fail causing damage to the SAC* illustrates a proposition (*cause*) which conveys explicit information on the causal relationship between other elementary facts.

As will be shown later, it is very helpful to introduce a taxonomy of elementary facts. Each of the elementary facts is a member of one of the following classes:

1. **Action** - Facts of this class describe actions performed by the operator. These can be divided into actions which influence the state of the model and whose consequences may be therefore simulated (subclass: Control, Repair) and actions which only provide information about other facts (subclasses Monitor).

1.1. **Control** - Actions leading to change of some parameters influencing ways in which equipment works. For example, setting switches to certain positions, pushing buttons, engaging clutches, etc. They are often used as parts of contexts both in the simulation queries and in the IF-THEN rules.

1.2. **Monitor** - Actions providing evidence for other facts: they usually wind up being related to other elementary facts by EVIDENCE links. They include perceptual acts dealing with what the operator saw, heard, felt, or smelled; inter-personal communications describing what was reported to the operator by other members of the crew; monitoring actions reporting what the operator measured or monitored and what the results of the measurements were; investigations and tests undertaken by the operator.

1.3. **Repair** - Actions dealing with fixing equipment parts, cleaning or replacing them, etc. This information is important to determine the final (i.e. after the report is dispatched) functionality of EUs.

2. **Event** - Facts of this class can be characterized by the question: "What happened to the

equipment?" At present we also treat as Events facts which more accurately would fall into a category of processes, e.g. *corrode*. Often a fact from this class is closely related to a State fact (see below) through a becoming act: Event-Fact-1 = (Become State-Fact-1 At-Time-1).

2.1. **Failure-Event** - Facts of this class should become linked into causal chains in the final results of understanding. The most usual causal relationships between facts of subclasses of this class are given below.

2.1.1. **Damage** - Facts of this class describe physical damage to EUs. The damage might be of two kinds: Fatal - a fatally damaged EU never works properly, e.g. *shear*; Sporadic - a sporadically damaged EU might sometimes work properly, e.g. *corrode*. Damage to an EU usually causes Malfunction of this EU.

2.1.2. **Disorder** - Facts of this class deal with abnormal values of WS parameters e.g. *lo pressure dropped*. A Disorder is usually caused by a Damage of some EU. A Disorder of a WS might lead to a Malfunction or/and Damage of some EU.

2.1.3. **Malfunction** - Facts of this class describe the improper behavior of EUs. A malfunctioning EU doesn't work as expected, e.g. *Drive shaft was found to rotate freely at the ssdg end*. There are two kinds of reasons for a malfunctioning EU: the EU is Damaged; or one or more of its input WSs have abnormal parameters (i.e. a Disorder is noticed).

2.2. **Normal-Op-Event** - Facts of this class describe events which usually happen when the equipment functions properly. They usually give information about the context of other facts or provide evidence for an assumption that some parts of the equipment function correctly.

2.3. **OK-Finding** - Facts of this class pertain either to EUs or to WSs. In the case of EUs they convey that these EUs either function correctly or are not damaged. In the case of WSs they convey that one of their parameters is as it normally should be. OK-Findings can be viewed as opposites to Damages, Disorders, and Malfunctions. They are important as a verification tool for hypotheses - their presence in the report makes some hypotheses invalid.

3. **Procedure** - Facts of this class are usually sequences of Control actions and Normal-Op-Events. Their role in understanding is to give temporal relations between some facts commonly mentioned in messages. They also provide script-type information about some common scenarios occurring during equipment operation, maintenance, and monitoring.

4. **State** - Facts stating that a particular feature of the domain equipment is true at some specific moment. The most interesting states are those which become true as results of some events; for example the state *broken tooth* is tied with the *break* event. This leads to a State-Event duality for such facts. Consider for example the sentence: *Lube oil pump was found to be seized*. Taken literally, it informs us of a certain state of the pump at the moment of the finding. However, from the viewpoint of understanding the whole report which includes this fact, it is more important to figure out when the event of seizure occurred, because then it is possible to include this fact into a causal chain and to assign to it a node in the event time graph. We have two subclasses which have their corresponding subclasses among Event facts:

4.1. **Failure-State** - Facts describing abnormal states of EUs, e.g. *wiped bearing*, *clogged filter* or abnormal WS parameters, e.g., *low lube oil pressure*.

4.2. **Normal-Op-State** - Facts describing desirable states of EUs, e.g. *Unable to start main propulsion gas turbine*. (meaning the inability of reaching the desirable state in which the turbine becomes started, i.e. running properly) or WS parameters, e.g. *Unable to maintain lube oil pressure to starting air compressor*. (meaning the inability of maintaining the desirable state in which the the pressure is in normal range).

Each action or event is associated with one point in the time graph (partial ordering); each state with two points, the beginning and end of the interval when the state was true.

7.4. Causal and Temporal Relationships

We have noted above the importance of linking events into causal chains. For some reports it is necessary to build more than just one such chain. Usually they overlap partially. For

example, the following CASREP

> While diesel was operating with SAC disengaged, the SAC LO alarm sounded. Believe the coupling from diesel to SAC lube oil pump to be sheared. Pump will not turn when engine jacks over.

has two causal chains: one involving the alarm, the other containing the engine jacking over. The shearing of the coupling is a common starting point for both these chains.

The most important reason for making causal and temporal relationships explicit during understanding is the observation that some of the information we considered important for understanding of CASREPs in section 7.1. can be extracted from causal chains. For example, they help to distinguish between primary and derivative problems as well as between intermediate and ultimate consequences of damages. Sometimes, especially in analyzing failure trends to discover possible design flaws, it may be very important - in cases when two or more damages were reported - to know whether one of them could have caused the other ones.

Building causal and temporal dependencies between reported facts reflects a significant aspect of understanding. We can view a CASREP as a story and following Schank's [1977] approach claim that an event in a CASREP is understood only when a plausible explanation for that event with respect to other events in the CASREP is found. The approach of causal and temporal chain building to understanding has been mainly applied to stories about personal encounters, for example Wilensky [1983]. The understanding process was based on a collection of rules capturing knowledge about beliefs and actions of people. For two given facts from a story, a causal link between them was established, if it was possible to construct a path which had the two facts as endpoints and possibly some hypothesized facts as intermediaries. The crucial feature of this path was that any two adjacent facts from it were related by causality expressed by some rule from the knowledge base. The solution we chose for analyzing CASREPs is an extension of this approach. In contrast to the knowledge about people's behavior which is characterized by a relatively large margin of uncertainty, the knowledge on which we base our understanding is in substantial part very precise: many of the events encountered in CASREPs can be simulated in a deterministic way using the PROTEUS equipment model. In the section describing the model we demonstrated that it is possible to propagate consequences of an event throughout the model to the point when a stable state is reached. It means that given two facts as candidates to be causally linked, we can simulate one of them and check whether the other can be confirmed in the newly established stable state of the model. One of the problems of finding causal links is to guess which pairs of elementary facts should be tried in order to avoid a huge number of queries. Most helpful here is the taxonomy of elementary facts into classes: only facts of certain classes may be linked into causal chains. Given two facts, one of the class OK-FINDING and another of the class DISORDER, we don't need to investigate whether there could be a causal relation between them. On the other hand, if the first fact were of class DAMAGE, then it is possible that these two facts are related by causality, whereby the probability of the DAMAGE fact having caused the DISORDER fact is higher than the probability of the reverse being true. Therefore, the first scenario is investigated first, and only if no confirmation is found is the other possibility considered. Another heuristic we apply for constructing useful simulation queries consists of grouping elementary facts from class Failure into sets corresponding to the media on which the damaged or malfunctioning EUs operate and which are affected by disorders. We can get the relevant EUs and WSs from the arguments of Damage, Malfunction, and Disorder facts, respectively. Each such set contains facts which are good first candidates for arguments of causal relationships. For more details see [Joskowicz et al, 1987].

The ultimate goal of this part of the understanding procedure is to include all elementary facts of some classes into some causal chain. When it is not possible to achieve this goal, attempts are made to hypothesize facts and treat them as additional elementary facts for the purpose of building more elaborate causal chains which would possibly include the "loose" facts. The hypotheses are based on more general knowledge about equipment, and especially its failures. This knowledge is organized as a set of rules of the form:

IF *fact-ante* IN CONTEXT OF {*fact-i*} THEN *fact-cons*

The above rule facts have the form of proposition-like patterns. An example is

IF *(Fail ?X)* IN CONTEXT OF {*(Part-Of ?X ?Y)*} THEN *(Fail ?Y)*

It is important to notice here that the inferred fact is only **possibly** true. A hypothesis is supported if it can be successfully used to link two original facts into a causal chain.

Another important structure we try to impose on elementary facts during discourse analysis deals with temporal relations. We distinguish between two such structures: **report time sequence**, i.e. the time when things were discovered, and **event time sequence**, i.e. the time when things happened. In most cases the information about the partial ordering provided by the Clause Semantics Module is not sufficient to build these structures completely. The remaining gaps are filled during the subsequent stages of analysis. The reason for distinguishing between the two times is the fact that they don't coincide. The two structures reflect two different perspectives: the first one describes actions from the maintenance point of view, the second from the causal point of view. Consider the following CASREP fragment: *Experienced loss of SAC lube oil pressure. [...] Drive shaft sheared all internal gear teeth from drive adapter hub*. The **finding** about the shearing occurred most probably **after** the **finding** about the loss of pressure (report time). On the other hand, the **shearing** itself most probably occurred **before** the loss of pressure (event time). The first ordering can be deduced from the sequence of sentences in the report: we can assume that time does not move backwards in narratives unless explicit time marker is provided, Allen [1983]. However, the report doesn't give any clue about the ordering in the event time; it can be deduced only after a causal relationship between the two events has been determined: we can assume that the causal chain provides a natural sequencing of events. Another example of information for building the event time graph are the operational characteristics of the domain equipment (for example, the compressor must be working before the starting of the turbine can be attempted).

7.5. Model Query Processor

In section 6.4 we showed how queries to the model were helpful for noun phrase analysis. Here we concentrate on model queries related to the discourse analysis. We differentiate between dynamic **simulation queries** and static **structural queries**.

The simulation queries have the form:

MQ-Simulation (Event-Prop, State-Prop, Context)

where *Event-Prop* and *State-Prop* are propositions corresponding to elementary facts of type EVENT and STATE respectively, and *Context* is an ordered list of propositions corresponding to elementary facts of type EVENT or ACTION. The possible results are either FALSE or TRUE. The interpretation of a simulation query is: "Does the simulation of the fact corresponding to *Event-Prop* in the *Context* lead to *State-Prop* becoming true."

Structural queries are used in IF-THEN rules for checking the context in which they may be applied. Most of the context constraints can be expressed by the predicate verification queries which are used for the noun phrase analysis (comp. section 6.4.3) but some additional query forms were required. One structural query which was introduced primarily for discourse analysis examines paths between model nodes. It has the form:

(MQ-Path From-Node To-Node &Optional From-Medium To-Medium Always-Medium)

From-Node and *To-Node* are two different nodes in the model. *From-Medium, To-Medium and Always-Medium* are optional arguments which in our domain take one of the values: OIL, AIR, or ROTATION. MQ-Path returns NIL if there is no path between the two nodes. Otherwise, it returns the first path found (there may be more!). The optional arguments put constraints on the links. If *From-Medium* is specified, the WS associated with *Link-1* must have the given medium as part of its description. If *To-Medium* is specified, the WS associated with *Link-n* must have the given medium as part of its description. If *Always-Medium* is specified, the WSs associated with all the links *Link-i* must have the given medium as part of their descriptions.

An example of an IF-THEN rule which uses MQ-Path queries in its CONTEXT part is:

```
IF (SeizedP ?X)
  IN CONTEXT OF
    {(GeneratorP ROTATION ?Y)
     (MQ-Path ?Y ?Z :Always-Medium ROTATION)
     (MQ-Path ?Z ?X :Always-Medium ROTATION)}
  THEN (Shear ?Z)
```

The interpretation of this rule is: "If an element is seized, then it is possible that an element lying on the path of rotary elements between a generator of the rotation and the seized element shears."

7.6. Report Summary

The last stage of the discourse analysis creates a report summary which reflects the purposes of analyzing CASREPs - they can be different depending on where and to whom the reports are dispatched (comp. section 7.2., above). The information which gets into the summary is extracted from the structure containing everything which PROTEUS was able to understand (the collection of elementary facts and their relationships). For each specific application there exists a different summary prototype designed in such a way that the most crucial information (from the perspective of a given application) will be extracted. For example, a summary might contain tables of damage, functionally impaired equipment, and corrective actions. Each damage entry would include the industrial name of the damaged element, the place of the element in the modeled equipment, the nature of the damage, the degree of certainty that the damage occurred, and the information on whether the damaged element had been fixed, replaced or left in place. The impaired equipment and corrective action entries would include similar details.

8. Equipment Model Editor

The PROTEUS system presented above, at its present stage of development, can only process reports about malfunctions of the starting air system. Considering that this system constitutes only a tiny portion of the entire range of shipboard equipment covered by CASREPs, we cannot claim at present to have a very practical text processing system.

If technical text understanding systems like PROTEUS, based on equipment simulation, are to be of any practical value, techniques must be provided to enable users other than the designers of such systems to tailor or expand them to new domains. The most difficult part of this task is building simulation models for the equipment in the new domains. Being aware of all this from the very beginning in our research, we have made many design decisions to facilitate the future development of a module for the computer-assisted design of new equipment models.

Even now creation of a model doesn't require specification of every slot in every single instance in the model. In the first place, many of the instance slot fillers are derived from information in the prototypes (see section 4.3). In addition, we have developed a quite elaborate initialization procedure whose task it is to build a fully-fledged model from specifications expressed in a more concise and natural way than meticulous instance definitions. In particular, we require at present that only instances describing EUs be given explicitly. Some slots of these instances contain information for the initialization procedure. The most notable example here are slots describing how the EU modeled by an instance is linked to other EUs on the same and on the immediately higher levels. This information is used by the initialization procedure to create instances of appropriate links. The information about links to be found in these slots is accompanied by information specifying the constant features of the WSs which are supposed to be associated with the links. The instances of these WSs are built automatically by the initialization procedure as well.

A very important resource at our disposal which can dramatically improve the model building process is our graphical display. The initialization procedure we have developed so far

can provide the basis for a powerful equipment model graphic editor. We have been aware of this possibility from the very beginning and have fashioned the graphical display with this potential application in mind. For example, all of the model network icons for nodes and links have been written into a regular grid which can be made visible on demand.

A graphical editor would allow us to replace the current approach of coding the individual EU instance specifications with an interactive session. At the beginning of the editor session an empty grid would be displayed along with a menu of icons for the prototype equipment like valves, pumps, filters, gears, etc. The user could then drag any of these icons into a chosen place of the grid. This done, the editor would involve the user in a menu-driven dialogue to capture the specific information needed to create an instance of the prototype corresponding to the chosen icon. In a similar fashion the user could specify all the links (i.e. dragging their icons into appropriate places and providing specific information by means of a dialog). The information necessary to complete descriptions of WS of some kind associated with links could be given only at the nodes which are sources of this kind of WS. The setting of nodes and links would be repeated until the user had completed an entire network. The user would then chose one of the displayed nodes for recursive refinement. This stepwise top-down process could continue until the entire model has been specified.

This scenario assumes that the whole model is built from instances of predefined prototypes. It is more realistic to assume that it will be possible to piece together only a certain part (although probably quite substantial) of a new equipment from predefined prototypes. Therefore, it will be necessary to facilitate defining new prototypes as well. This will probably require a more thorough knowledge about the system on the part of the user.

9. Implementation

The PROTEUS system has been substantially implemented and debugged and has been publically demonstrated operating on a small set of actual CASREPs. Of the components described in detail above: the equipment model has been fully implemented as described. The noun phrase analyzer has been fully implemented except for the mechanism described in the last paragraph of section 6.3.1 and the contributions of clause semantics and discourse analysis to reference resolution. For discourse analysis we have implemented to date only the basic causal analysis without hypothesis generation.

10. Conclusion

CASREPs are written on board Navy ships and sent to various Navy commands for collection and analysis. PROTEUS has been designed for use by the recipients of these messages, in order to automate the message processing. In these concluding remarks we want to investigate an attractive extension to this application.

As we have seen in this paper, the approach we have taken is based on a rather detailed simulation of the equipment which is the subject of the CASREPs to be analyzed. The simulation model is the "world" in which we interpreted CASREPs texts. Looking at reported facts in the context of this world allowed us to solve some difficult understanding problems, most notably identifying concepts referred to in the reports and finding coherence relationships between their individual sentences. This was possible because the simulation model proved to be a rich source of information about the domain as well as a good inference mechanism. It was also a useful tool for checking hypotheses. In effect, the model together with the query processor operating on it could be justifiably referred to as a small expert system.

We believe that expanding PROTEUS to a powerful interactive diagnostic assistant would be a logical consequence of the approach we took to language understanding. Then it would be possible to deploy PROTEUS systems not only where the CASREPs are collected but directly on Navy ships. To see why this should be desirable let us make some general comments on CASREPs. Reading them we got the feeling that the degree of effort (later reported) and expertise involved in attempts to locate reasons for problems differed greatly

from case to case. At one extreme there are cases when nothing was done to even identify the nature of the problem. On the other hand there are quite long reports (too long to be quoted here) which describe successful diagnostic sessions leading to the full identification of the causes of failure. In between are CASREPs which describe symptoms of problems and then:

- only speculate on the possible damage which could explain these symptoms (*It is likely the LO pump has sheared.*), or
- give indirect evidence without indicating any damage (*Start air pressure dropped below 30 psig during monitoring of 1A turbine. Oil is discolored and contaminated with metal.*), or
- postpone the diagnosis until more tests have been conducted (*Retained oil sample and filter element for future analysis.*), or
- give no reasons for the occurrence of the symptoms at all (*During routine start of main propulsion gas turbine, SAC air pressure decreased rapidly to 5.74 psi resulting in an aborted engine start. Exact cause of failure unknown.*).

These wide variations suggest that, if only the ship technicians were better experts, the level of diagnosis would have been deeper and the CASREPs correspondingly more informative.

Because it is unrealistic to expect that all technicians posess the same high level of expertise, it could be helpful to have an interactive diagnostic assistant on the ship which could lead a relatively inexperienced technician through a diagnostic session. This guidance would take the form of suggestions for tests to be performed and requests to the technician for more diagnostic information. The success of such a system would depend a lot on the ease with which the initial symptoms could be described and the quality of the subsequent interactions. Because of the wide variety of possible symptoms (as evidenced by the CASREPs), a combination of a rich natural language capability and a graphic interface seems called for. We believe therefore that PROTEUS, with its integrated language understanding, graphic interface, and detailed equipment model, is particularly well suited to be the base for such a system.

Acknowledgments

This paper is based upon work supported by the Defense Advanced Research Projects Agency under Contract N00014-85-K-0163 from the Office of Naval Research, and by the National Science Foundation under grant DCR-85-01843. We wish to acknowledge the contributions of: Leo Joskowicz, who developed some of the causal analysis procedures and assisted in the final preparation of this manuscript; John Sterling, who developed the current implementation of the noun phrase analyzer; Ngo Thanh Nhan, who developed the grammar and implemented the clause analyzer; and Michael Moore, who integrated the components on the Symbolics.

References

[Allen 1983] Allen, J. Maintaining knowledge about temporal intervals, *CACM*, **26**, 1983, 832-843

[Bobrow 1977a] Bobrow, D. and Winograd, T. An overview of KRL, a Knowledge Representation Language, *Cognitive Science*, 1977, 1-46

[Bobrow 1977b] Bobrow, D., Kaplan, R.M., Kay, M., Norman, D.A., Thompson H. and Winograd, T. GUS, a frame-driven dialog system, *Artificial Intelligence*, 1977, 155-173

[Bobrow 1985] Bobrow, D.G. (ed) Qualitative Reasoning about Physical Systems, The MIT Press, 1985

[Bylander 1985] Bylander, T., Chandrasekaran, B. Understanding behavior using consolidation, *Proc. Ninth Int. Joint Conf. on Artificial Intelligence*, Los Angeles, 1985, 450-454

[Charniak 1978] Charniak, E. On the use of framed knowledge in language comprehension, *Artificial Intelligence*, 1978, 225-265

[Finin 1986] Finin, T. Nominal compounds in a limited context, in: *Analyzing Language in Restricted Domains*, R. Grishman and R. Kittredge (eds), Lawrence Erlbaum, 163-173

[Forbus 1984] Forbus, K.D. Qualitative process theory, *Artificial Intelligence*, 1984, 85-168

[Grishman 1986] Grishman, R., Ksiezyk, T. and Nhan, N.T. Model-based analysis of messages about equipment. PROTEUS Project Memo #2, CSD, NYU

[Hirschman 1986] Hirschman, L. Personal communication.

[Hollan 1984] Hollan, J., Hutchins, E. and Weitzman, L. STEAMER: an interactive inspectable simulation-based training system, *AI Magazine*, Summer 1984, 15-27

[Joskowicz 1987] Joskowicz, L., Grishman, R. and Ksiezyk, T. Finding causal and temporal relations in equipment failure messages, PROTEUS Project Memo #8, CSD, NYU

[de Kleer 1984] de Kleer, J., Brown, J.S. A qualitative physics based on confluences, *Artificial Intelligence*, 1984, 7-83

[Ksiezyk 1987] Ksiezyk, T., Grishman, R. and Sterling, J. An equipment model and its role in the interpretation of noun phrases, *Proc. Tenth Int. Joint Conf. on Artificial Intelligence*, Milano, 1987

[Kuipers 1984] Kuipers, B. Commonsense reasoning about causality: deriving behavior from structure, *Artificial Intelligence*, 1984

[Lehnert 1982] Lehnert, W.G. Plot units: a narrative summarization strategy, in: *Strategies for Natural Language Processing*, W.G. Lehnert and M. Ringle (eds), Lawrence Erlbaum

[Marsh 1984] Marsh, E., Hamburger, H. and Grishman, R. A production rule system for message summarization, *Proc. of the Nat. Conf. on Artificial Intelligence*, Palo Alto, 1984, 243-246

[Norvig 1983] Norvig, P. Frame activated inferences in a story understanding program, *Proc. Eighth Int. Joint Conf. on Artificial Intelligence*, 1985, 624-626

[Palmer 1986] Palmer, M., Dahl, D., Schiffman, R., Hirschman, L., Linebarger, M. and Dowding, J. Recovering implicit information, *24th Annual Meeting of the ACL*, 1986, 10-19

[Rumelhart 1977] Rumelhart, D.E. Understanding and summarizing brief stories, in: *Basic processing in Reading, Perception, and Comprehension*, D. Laberge and S. Samuels (eds), Hillsdale, 1977

[Sager 1978] Sager, N. Natural language information formatting: the automatic conversion of texts to a structured data base, in: *Advances in Computers* 17, M.C. Yovits (ed), Academic Press, 1978

[Schank 1977] Schank, R. and Abelson, R. Scripts, Plans, Goals, and Understanding, Lawrence Erlbaum, 1977

[Wilensky 1983] Wilensky, R. Planning and Understanding. A Computational approach to Human Reasoning, Addison-Wesley, 1983

END
DATE
FILMED
3-88
DTIC